Méthode de faisceaux désagrégée avec gradients creux

Claude Lemaréchal, Claudia Sagastizábal

INRIA

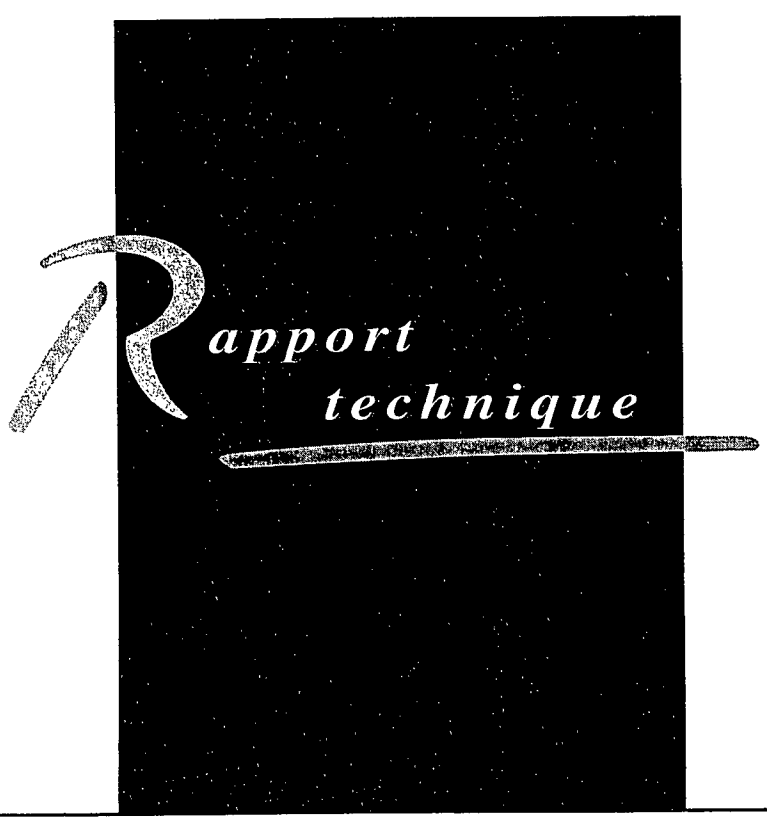
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Méthode de faisceaux désagrégée avec gradients creux

Claude Lemaréchal - Claudia A. Sagastizábal

N° 0216
Janvier 1998

THÈME 4



INRIA
ROCQUENCOURT

Méthode de faisceaux désagrégée avec gradients creux

Claude Lemaréchal , Claudia A. Sagastizábal

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Promath

Rapport technique n 0216 — Janvier 1998 — 19 pages

Résumé : L'optimisation de la production d'électricité, résolue par relaxation lagrangienne, résulte pour la phase de coordination en un problème non différentiable de grande taille. Les temps de calcul impliqués nécessitent un algorithme de coordination très performant, qui ne doit pas requérir trop de résolutions des problèmes locaux. Pour cela, on tire parti de la structure additive de la fonction duale: chaque agent local peut donner lieu à sa propre linéarisation par plans sécants, ce qui raffine l'approximation de la fonction duale.

Le présent rapport décrit l'implémentation de cette technique dans une méthode de faisceaux récente. Les performances du code résultant sont illustrées sur divers problèmes de gestion de la production. Ce travail a fait l'objet d'un contrat entre EdF et l'Inria.

Mots-clé : gestion de la production électrique, optimisation, relaxation lagrangienne, méthode de faisceaux, accélération d'algorithmes

(Abstract: pto)

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

Disaggregated bundle methods with sparse gradients

Abstract: The unit-commitment problem, solved by Lagrangian relaxation, results for the coordination phase in a large-scale nonsmooth optimization problem. The computing times involved require a very efficient coordination algorithm, which must not need too many resolutions of the local problems. For this, advantage is taken from the additive structure of the dual function: each local agent can yield its own cutting plane linearization, thereby refining the approximation of the dual function.

The present report describes the grafting of this technique into a recent bundle method. The behaviour of the resulting program is illustrated on various unit-commitment problems. This work was the object of a contract between the French Electricity Board (EdF) and Inria.

Key-words: unit-commitment problem, optimization, Lagrangian relaxation, bundle method, acceleration of algorithms

1 Synthèse

Ce texte constitue le rapport de fin de contrat Inria-EdF n° R31/1J9073/ER275. L'objet de ce contrat était de tester la possibilité d'accélérer la méthode de faisceaux, développée dans le cadre du contrat CERD R31/1J3669/ER238. Pour cela, il s'agissait d'implémenter une variante dite de désagrégation, ayant déjà fait ses preuves pour la méthode de Dantzig-Wolfe pure. Il s'agissait en particulier d'appliquer cette variante à la décomposition croisée, seule technique viable dans un problème de production prenant le réseau en compte.

Le résultat du présent contrat est tout d'abord un affinage, sous l'angle de la robustesse et de la cohérence, du code de faisceaux livré à l'issue du contrat CERD R31/1J3669/ER238. Par ailleurs, une version désagrégée est maintenant opérationnelle et donne satisfaction sur la décomposition spatiale classique. Les temps de calcul sont divisés en gros par 5, pour des modèles stochastiques ayant jusqu'à 10^3 variables duales.

En revanche, il n'est pas encore possible de conclure quant aux gains apportés par la désagrégation, appliquée à la décomposition croisée du parc français. Pour implémenter une telle technique, il nous a fallu surmonter de substantielles difficultés informatiques, provoquées par la grande taille inhérente à ce mode de décomposition. Plus précisément, ces difficultés ont été de deux types:

- Concernant l'encombrement-mémoire: exploitation du creux apporté par la décomposition croisée.
- Concernant les temps de calcul:
  - . introduction du creux dans le code `q2edf` de K.C. Kiwiel (Interfaces Institute, Varsovie), chargé de résoudre les problèmes quadratiques apparaissant dans la méthode de faisceaux,
  - . filtrage des informations redondantes en provenance des problèmes locaux.

Cela étant fait, il reste à valider le code ainsi produit par des expérimentations supplémentaires sur des modèles de décomposition croisée.

2 Le problème posé

Nous commençons par décrire dans ses grandes lignes la relaxation lagrangienne appliquée à un problème décomposable. La gestion de la production se prête à cette technique, nous exposons en §2.2 les trois problèmes de ce type traités par la suite.

2.1 Relaxation lagrangienne

Soit un problème d'optimisation décomposable, tel que la gestion de la production, formulé abstraitement comme suit:

$$\left\{\begin{array}{l} \min \sum_{\ell\le\mathcal{L}} f_\ell(z_\ell) =: f(z)\,, \\ z_\ell \in \mathcal{Z}_\ell \quad \text{pour } \ell \le \mathcal{L}\,, \quad \text{i.e. } z \in \mathcal{Z}\,, \\ \sum_{\ell\le\mathcal{L}} h_\ell(z_\ell) =: h(z) = 0\,. \end{array}\right. \tag{1}$$

La présence de cette structure décomposable est importante pour la suite. Les indices ℓ repèrent les *agents locaux*; dans le problème qui nous occupe, ce seront essentiellement des unités de production, $\mathcal{Z}_\ell$ représentant les contraintes technologiques de chaque unité (contraintes "dures"). Les contraintes $h(z) = 0$ introduisent donc un couplage entre les agents, et c'est ce qui fait la difficulté essentielle du problème.

Supposant que la contrainte couplante $h := \sum h_\ell$ est à valeurs dans R^n – dans lequel $\langle\cdot,\cdot\rangle$ note le produit scalaire habituel –, on introduit le vecteur de *variables duales* ou prix $\lambda \in \mathrm{R}^n$ (le même pour tous les agents) et le lagrangien

$$L(z,\lambda) := f(z) - \langle\lambda, h(z)\rangle = \sum_{\ell\le\mathcal{L}} L_\ell(z_\ell,\lambda)\,.$$

Nous avons conservé la structure additive du problème initial:

$$L_\ell(z_\ell,\lambda) := f_\ell(z_\ell) - \langle\lambda, h_\ell(z_\ell)\rangle$$

car c'est important pour la suite. On définit maintenant la *fonction duale*

$$\Theta(\lambda) := \inf_{z\in\mathcal{Z}} L(z,\lambda) = \sum_{\ell\le\mathcal{L}} \Theta_\ell(\lambda)$$

qui présente encore la même structure additive:

$$\Theta_\ell(\lambda) := \inf_{z_\ell\in\mathcal{Z}_\ell} L_\ell(z_\ell,\lambda) = \inf_{z_\ell\in\mathcal{Z}_\ell} [f_\ell(z_\ell) - \langle\lambda, h_\ell(z_\ell)\rangle]\,. \tag{2}$$

En résumé, nous nous intéressons d'une façon générale aux problèmes possédant une structure additive comme en (1). Nous désirons profiter de cette structure, dont l'effet est de décomposer le calcul de Θ en sous-problèmes $(2)_\ell$: ces derniers sont donc supposés "beaucoup plus faciles" que (1). L'outil pour résoudre (1) via (2) est alors la dualité, dont nous rappelons les résultats essentiels:

- La résolution de (1) via (2) passe par le *problème dual*

$$\max_{\lambda\in\mathrm{R}^n} \Theta(\lambda) = \max_{\lambda\in\mathrm{R}^n} \sum_{\ell\le\mathcal{L}} \Theta_\ell(\lambda)\,. \tag{3}$$

- Chaque Θ_ℓ est une fonction concave de λ; Θ est donc également concave, le problème dual est bien posé.
- Si $z_\ell(\lambda)$ est une solution du ℓ^e problème local (2) (i.e. $L_\ell(z_\ell(\lambda),\lambda) = \Theta_\ell(\lambda)$), alors la valeur correspondante de la contrainte donne un sous-gradient de la fonction convexe $-\Theta_\ell$:

$$\Theta_\ell(\mu) \le \Theta_\ell(\lambda) - \langle h_\ell(z_\ell(\lambda)), \mu-\lambda\rangle \quad \text{pour tout } \mu\in\mathrm{R}^n. \tag{4}$$

- L'approche duale n'est qu'heuristique: on n'obtient en général pas une solution de (1) à partir d'une solution duale (*saut de dualité*).

Remarque Les propriétés suivantes complémentent les résultats ci-dessus:

- Si les contraintes couplantes dans (1) étaient des inégalités $h(z)\ge 0$, le problème dual (3) aurait les contraintes $\lambda\ge 0$.
- Sauf cas pathologiques qui peuvent être négligés ici, le gradient $\nabla\Theta_\ell(\lambda) = -h_\ell(z_\ell(\lambda))$ existe chaque fois que $(2)_\ell$ a une solution *unique* $z_\ell(\lambda)$. Sinon, $h_\ell(z_\ell(\lambda))$ est mal défini: a priori, $-\Theta_\ell(\lambda)$ a au moins autant de sous-gradients qu'il y a de solutions optimales dans $(2)_\ell$. Moralité: Θ_ℓ est concave mais en général non différentiable.
- La qualité de l'heuristique peut être mesurée a posteriori: $z(\lambda)$ est solution de

$$\min_{z\in\mathcal{Z}} f(z) \quad \text{sous les contraintes perturbées} \quad h(z) = \bar{h}\,,$$

où $\bar{h} := h(z(\lambda))$. On obtient donc une solution approchée de (1) si $h(z(\lambda))$ est voisin de 0, exacte si $h(z(\lambda)) = 0$.
- La propriété $h(z(\lambda)) = 0$ (qui fournit donc une solution primale) entraîne nécessairement que $0\in\partial\Theta(\lambda)$, i.e., λ résout (3). Sans entrer dans les détails techniques, cette propriété est accessible lorsque (1) est un problème d'optimisation convexe, ou encore lorsqu'une solution duale $\bar{\lambda}$ est telle que chaque problème $(2)_\ell$ a une solution unique. Il n'est pas exagéré de dire qu'en pratique, l'équation $h(z(\lambda)) = 0$ n'a de solution que dans ces deux cas.

Pour un exposé plus complet de cette théorie, voir par exemple [?, Chap. 8], ou [?, Chap. XII]. □

En résumé, la relaxation lagrangienne est une approche (heuristique) pour résoudre (1) en le ramenant à un problème d'optimisation non différentiable. L'objet du présent contrat était de tester une variante de la méthode de faisceaux (exposée en §3.3) pour résoudre de tels problèmes.

Comme d'habitude en optimisation, le logiciel correspondant se compose de deux parties bien distinctes. Pour reprendre la terminologie Modulopt, ces deux parties sont:

- l'*optimiseur*, qui se charge d'itérer sur les λ dans le but de résoudre le problème dual; il fera l'objet de la §3;
- le *simulateur* qui, connaissant λ donné par l'optimiseur, se charge de résoudre les problèmes locaux (2) pour répondre les $\Theta_\ell(\lambda)$ et les $h_\ell(z_\ell(\lambda))$ à l'optimiseur; nous en donnons des exemples dans la §2.2 qui suit.

2.2 Les problèmes considérés

Dans cette section, nous présentons trois problèmes de gestion de la production, traités dans le cadre du présent contrat.

2.2.1 Le problème N comme "Naturel"

Considérons le problème de production journalière déterministe pour un parc comportant les unités de production $i\le\mathcal{I}$. La journée d'étude est divisée en T périodes de temps (en pratique: 48 demi-heures). Appelant

p_i^t la production de l'unité i pendant la période t. nous noterons $p_i = (p_i^1, \ldots, p_i^T)$ le vecteur-production de l'unité i. et $p^t = (p_i^t)_{i \leq \mathcal{I}}$ le vecteur-production pendant la période t. Le problème à résoudre est alors

$$\begin{cases} \min \sum_{i \leq \mathcal{I}} c_i(p_i) \,. \\ p_i \in \mathcal{P}_i \,. \quad \text{pour } i \leq \mathcal{I} \,. \\ p^t \in \mathcal{D}^t \,. \quad \text{pour } t = 1, 2, \ldots, T \,. \end{cases} \tag{5}$$

Ici. $c_i : \mathrm{R}^T \to \mathrm{R}$ est le coût de production de l'unité i et $\mathcal{P}_i$ décrit les contraintes dynamiques caractérisant cette même unité. Le domaine $\mathcal{D}^t \subset \mathrm{R}^I$ décrit les contraintes statiques devant être satisfaites pendant la période t. Dans les problèmes retenus. ne figurent ni les contraintes de réseau. ni la réserve tournante: seule figure la satisfaction de la demande. soit une contrainte scalaire pour chaque période:

$$p^t \in \mathcal{D}^t \qquad \text{signifie } (p_i^t \geq 0 \text{ et}) \quad \sum_{i \leq \mathcal{I}} p_i^t = d^t \,. \tag{6}$$

Les contraintes $p \geq 0$ sont censées figurer dans $\mathcal{P} := \prod_{i \leq \mathcal{I}} \mathcal{P}_i$. il est commode de les répéter ici.

Remarque La formulation (6) en égalités correspond à la présence d'un groupe fictif. disons le n°0. Il joue le rôle de variable d'écart. sans coût de production ($c_0 \equiv 0$) et avec des contraintes dynamiques du type $\mathcal{P}_0 = \{p_0 : 0 \leq p_0^t \leq \bar{p}\}$. Les contraintes statiques sont plutôt $\sum_{i=1}^{\mathcal{I}} p_i^t - p_0^t = d^t$. de sorte qu'en fait on a dans (6) $\sum_{i=1}^{\mathcal{I}} p_i^t \geq d^t$.

Par ailleurs. rappelons ici que l'étude concernait la faisabilité d'une certaine approche (la méthode de faisceaux désagrégée). Les contraintes de réserve tournante sont formellement identiques aux contraintes de demande. les négliger n'infirme en rien le modèle. Négliger les contraintes de réseau est plus grave. nous l'avons fait surtout par manque de temps: elles auraient considérablement compliqué les simulateurs. et nous avons préféré nous attacher à d'autres difficultés. indépendantes de la forme des contraintes statiques: ces difficultés seront décrites en §3.3.2. □

Supposons pour l'instant que $d^t = 0$. Le problème N consiste alors tout simplement à identifier (1) et (5)-(6) de façon naturelle. c'est-à-dire faire $\ell = i$. $\mathcal{L} = \mathcal{I}$. $z = p$ (donc $z_\ell = p_i \in \mathrm{R}^T$). $f = c$. $\mathcal{Z} = \mathcal{P}$. $h_\ell(z_\ell) = p_\ell$ (mais $h_0(z_0) = -p_0$) et $n = T$. Les problèmes locaux s'obtiennent en recopiant (2) "mot à mot":

$$\Theta_i(\lambda) := \inf_{p_i \in \mathcal{P}_i} L_i(p_i, \lambda) = \inf_{p_i \in \mathcal{P}_i} \left[c_i(p_i) - \sum_{t=1}^{T} \lambda^t p_i^t \right] . \tag{7}$$

Chacun d'entre eux consiste à optimiser le planning de l'unité i. compte tenu des prix λ^t rétribuant la participation de cette unité pour satisfaire la demande globale.

Pour $d^t \neq 0$. on obtient par translation la fonction duale $\Theta(\lambda) = \sum_{i \leq \mathcal{I}} \Theta_i(\lambda) - \sum_{t=1}^{T} \lambda^t d^t$. ce qui laisse inchangés les problèmes locaux.

Remarque Le problème fictif $(2)_0$. qui s'écrit

$$\inf_{p_0 \in \mathrm{R}^T} \sum_{t=1}^{T} \lambda^t p_0^t \qquad 0 \leq p_0^t \leq \bar{p} \,.$$

a une solution $p_0^t(\lambda)$ qui vaut soit 0. soit $\bar{p}$. suivant le signe de λ^t. La fonction duale correspondante est $\Theta_0(\lambda) = \sum_t \bar{p} \min\{0, \lambda^t\}$. Autrement dit. ce groupe fictif résulte dans l'espace dual en une pénalisation des contraintes duales $\lambda_t \geq 0$. qui apparaîtraient si la contrainte dans (6) était une inégalité: et $\bar{p}$ joue ainsi le rôle de coefficient de pénalité. □

Dans cette formulation. le problème dual consiste à maximiser une fonction Θ concave et non différentiable. dépendant de $n = T = 48$ variables. et somme de $\mathcal{I}$ fonctions duales locales Θ_i. Nous avons limité notre étude de ce problème à une maquette synthétique. comportant soit $\mathcal{I} = 10$. soit $\mathcal{I} = 100$ unités en production. toutes thermiques (simplifiées). et divers jeux de données (différant surtout par les demandes). Cette maquette nous est utile pour dégrossir nos modifications sur le code de faisceaux faisant l'objet du présent contrat:

- Du fait de sa simplicité. le simulateur peut être exploité sur n'importe quel site. en particulier à l'Inria.
- En jouant sur les demandes d^t. on peut varier la difficulté du problème. notamment fabriquer un domaine réalisable $\mathcal{P} \cap \mathcal{D}$ difficile à atteindre.

- Chaque problème (2) est purement en nombres entiers. Chaque fonction duale Θ_i est donc linéaire par morceaux, ce qui isole l'une des caractéristiques principales de l'optimisation non différentiable. De plus, les problèmes (2) se prêtent aisément à une résolution exacte; il n'en serait pas de même avec des groupes hydrauliques.
- Du fait des dimensions réduites, le simulateur consomme très peu de temps CPU. Cela permet de multiplier les tests, qui se font en temps réel à l'écran, et non pas en batch pendant la nuit.

Remarquer par ailleurs qu'un problème plus complet résulterait en des problèmes locaux plus nombreux et plus complexes, mais laisserait inchangée la dimension (modérée) de l'espace dual. Il n'en serait pas de même si le réseau était pris en compte.

2.2.2 Le problème X comme "croisé"

On vient de voir que le problème dual aurait un nombre de variables gigantesque si les contraintes de réseau étaient prises en compte. Prenons un exemple: une modélisation raisonnable du réseau français peut comporter environ 400 arcs, avec une contrainte de capacité pour chaque arc; le problème dit de *sécurité* $n-1$ (prévoyant le cas où l'un quelconque des arcs est indisponible) résultera alors en $400 \times 400 = 10^5$ contraintes.

Le schéma de *décomposition croisée* ci-dessous, qui apparaît comme une variante du schéma N pour résoudre le même problème (5), supprime cette difficulté. Dédoublons les variables p dans (5) pour obtenir le problème équivalent, dans lequel les variables sont p_i^t et q_i^t:

$$\left\{\begin{array}{l} \min \sum_{i\le\mathcal{I}} c_i(p_i)\,, \\ p_i \in \mathcal{P}_i \quad \text{pour } i \le \mathcal{I}\,, \\ q^t \in \mathcal{D}^t \quad \text{pour } t = 1,2,\ldots,T\,, \\ p = q \quad (\text{ou } p_i^t = q_i^t \text{ pour } i \le \mathcal{I} \text{ et } t = 1,2,\ldots,T)\,. \end{array}\right. \tag{8}$$

La décomposition croisée consiste alors à appliquer la relaxation lagrangienne de la §2.1 en dualisant les contraintes $p = q$. Sachant que $p = (p_i)$ et $q = (q^t)$, on fait donc l'identification $z = (p,q)$ et $h(z) = p - q$. Les contraintes "dures" sont maintenant $z = (p,q) \in \mathcal{Z} = \mathcal{P} \times \mathcal{D}$, où $\mathcal{D} := \prod_{t=1}^T \mathcal{D}^t$. Le lagrangien est

$$L(p,q;\mu) = f(p) - \langle \mu, p-q\rangle = \sum_{i\le\mathcal{I}} c_i(p_i) - \sum_{i\le\mathcal{I}} \sum_{t=1}^T \mu_i^t (p_i^t - q_i^t)\,.$$

On a appelé μ les variables duales pour bien marquer qu'elles n'ont rien à voir avec λ; elles ne varient plus dans R^T mais dans $\mathrm{R}^{\mathcal{I}} \times \mathrm{R}^T$; ici aussi, nous noterons $\mu_i = (\mu_i^1,\ldots,\mu_i^T)$ et $\mu^t = (\mu_i^t)_{i\le\mathcal{I}}$.

La structure additive du lagrangien est maintenant:

$$L(p,q,\mu) = \sum_{i\le\mathcal{I}} L_i(p_i,\mu_i) + \sum_{t=1}^T L^t(q^t,\mu^t)\,,$$

avec

$$L_i(p_i,\mu_i) = c_i(p_i) - \sum_{t=1}^T \mu_i^t p_i^t \qquad \text{et} \qquad L^t(q^t,\mu^t) = \sum_{i\in\mathcal{I}} \mu_i^t q_i^t\,.$$

Sa minimisation implique de résoudre deux types de problèmes. Il y a d'une part un problème pour chaque unité de production:

$$\Theta_i(\mu_i) := \inf_{p_i\in\mathcal{P}_i} L_i(p_i,\mu_i) = \inf_{p_i\in\mathcal{P}_i} \left[c_i(p_i) - \textstyle\sum_{t=1}^T \mu_i^t p_i^t\right],$$

rigoureusement identique à son homologue du problème N (à ceci près que les unités reçoivent des prix μ_i différents). Il y a d'autre part un problème pour chaque pas de temps:

$$\Theta^t(\mu^t) = \inf_{q^t} \sum_{i\in\mathcal{I}} \mu_i^t q_i^t \qquad \text{sous les contraintes } (q_i^t \ge 0 \text{ et}) \quad \sum_{i\le\mathcal{I}} q_i^t = d^t\,,$$

qui est un simple problème de sac à dos continu. Noter que, si $\mathcal{D}$ contenait les contraintes de réseau, chaque problème en q ci-dessus serait un problème de réseau ordinaire (statique, à l'instant t).

Remarquer le creux du problème dual: chaque fonction Θ_i [resp. Θ^t] ne dépend que de μ_i [resp. μ^t]; cela est important pour la suite.

Contrairement au problème N, nous avons considéré ici un problème réel, comportant l'ensemble du parc français sur une certaine journée. Les dimensions sont alors $\mathcal{I} = 140$ unités, donc $\mathcal{L} = 140 \div 48 = 188$ problèmes locaux; quant au nombre de variables duales, il vaut $n = 10848$ (qui est supérieur à $140 \times 48 = 6720$ car en fait, pour certains groupes, p est seulement le résultat "aggloméré" de plusieurs variables).

2.2.3 Le problème S comme "Stochastique"

Le troisième problème considéré est formellement identique à N, mais correspond à une application assez différente. La période $1, \ldots, T$ n'est plus une journée de 48 demi-heures mais une semaine divisée en 56 tranches de 3 heures. Alors, les demandes d^t ne sont pas connues: elles dépendent significativement de la météorologie (surtout de la température). Toutefois, l'évolution du temps n'est pas un phénomène complètement aléatoire: en fait, les organismes de prévision météorologique sont capables[1] de fournir un certain nombre, relativement réduit, de *scénarios* de température, et donc de demande.

Dans ces conditions, le problème S est le suivant.

- A partir d'une valeur d à l'instant t, la demande ne peut évoluer que vers un certain nombre de valeurs à l'instant $t+1$, disons $d_1, \ldots, d_k$, avec les probabilités $\pi_1, \ldots, \pi_k$. Noter que k et les d_j, π_j dépendent évidemment de t, mais aussi de d: certaines situations météorologiques sont plus instables que d'autres: on a souvent $k = 1$ et rarement $k > 2$.
- De façon récursive, la demande peut donc prendre à l'instant t un nombre fini de valeurs, affectées chacune d'une certaine probabilité. Pour un t donné, ces probabilités se somment à 1.
- Il est alors commode de représenter toutes ces valeurs sur un graphe, qui est en fait un arbre:
  - sa racine est en $t = 1$ (où d^1 est connue);
  - ses feuilles sont en $t = T$;
  - ses noeuds sont classés par "dates", qui sont leurs distances à la racine;
  - les noeuds à la date t représentent l'ensemble des demandes possibles à l'instant t;
  - une branche joignant deux noeuds aux dates respectives t et $t+1$ représente l'évolution de la demande entre les deux valeurs correspondantes;
  - à chaque feuille correspond un scénario: l'évolution de la demande de 1 à T est repérée par le chemin (unique!) joignant la racine et la feuille en question.
- Soit N le nombre de noeuds (le nombre total de valeurs que peut prendre la demande à tous les instants). A chaque noeud ν sont affectés trois nombres:
  - sa date t^ν,
  - d^ν, qui est l'état de la demande (parmi d'autres possibles à la même date t^ν)
  - π^ν, qui est la probabilité de cet état.
- On veut alors minimiser l'espérance du coût de production, tout en respectant les contraintes dynamiques de chaque unité, et en satisfaisant la demande quoi qu'il arrive (i.e. en chaque noeud de l'arbre).

En résumé, soit p_i^ν le niveau de production de l'unité i au noeud ν, et soit $c_i^\nu(p_i^\nu)$ le coût de production correspondant. On doit résoudre

$$\begin{cases} \min \sum_{\nu=1}^{N} \pi_\nu \sum_{i \le \mathcal{I}} c_i^\nu(p_i^\nu) \\ p_i \in \mathcal{P}_i \quad \text{pour } i \le \mathcal{I}, \\ \sum_{i \le \mathcal{I}} p_i^\nu = d^\nu \quad \text{pour } \nu = 1, \ldots, N. \end{cases}$$

Intervertissant si nécessaire les sommes sur ν et i, on voit que ce problème est formellement identique à (5)-(6) (nous n'avons pas répété les contraintes $p \ge 0$).

Remarque Mis à part un changement de notation, il y a essentiellement deux différences.

- Les dimensions: T est changé en N, qui peut être substantiellement plus grand en période d'instabilité météorologique. Ceci augmente aussi bien le nombre N de contraintes couplantes que le nombre $\mathcal{I} \times N$ de variables primales.
- Les commandes: chaque p_i^t est remplacé par la production moyenne de l'unité i à l'instant t, à savoir: $\sum_\nu \pi^\nu p_i^\nu$, où la somme porte sur l'ensemble des noeuds à la date t. □

1. ou le seront dans un proche avenir. Mentionnons ici que la plupart de ces organismes (dont Météo France et le Centre Européen en Angleterre) utilisent de façon cruciale `m1qn3`, un code d'optimisation développé par le Projet Promath de l'Inria.

Ce problème se prête lui aussi à chacune des deux décompositions lagrangiennes déjà vues. Nous avons choisi la première, procédant comme en §2.2.1, mutatis mutandis: dans (7), t et T deviennent ν et N, les $c_i(p_i)$ deviennent des coûts moyens comme dans la remarque ci-dessus.

Nous avons considéré trois jeux de données différents, comportant respectivement 312, 760 et 1016 noeuds (i.e. variables duales λ^ν) et $\mathcal{I} = 123$ groupes (tous thermiques). Par ailleurs, pour diversifier l'influence de $\mathcal{L}$ dans la résolution de (1), nous ne nous sommes pas contentés de faire la simple identification $\mathcal{L} = \mathcal{I} = 123$, $\ell = i$. Nous avons aussi pris $\mathcal{L} = 1$ (on ne tient aucun compte de la structure additive) et $\mathcal{L} = 13$ (12 paquets de 10 unités et un paquet de trois).

3 Résolution du problème dual

Puisque la relaxation lagrangienne ramène le problème posé à la maximisation de la fonction duale, qui est concave, cette section est consacrée à l'optimisation non différentiable. Nous commençons par quelques rappels sur ses principes, puis nous exposons en §3.3 la méthode de faisceaux désagrégée, objet du présent contrat. Nous adoptons les notations de la §2.1, sachant qu'elles peuvent être adaptées à chacun des problèmes considérés, comme il a été vu en §2.2.

Le point de vue adopté dans toute cette section est celui de l'optimiseur (l'algorithme dual), qui ignore tout du contenu de la §2.2: les problèmes locaux (2) ne sont ici qu'autant de *boîtes noires* (les simulateurs), prenant les variables duales λ en entrée et répondant en sortie: les valeurs duales Θ_ℓ et les valeurs correspondantes des contraintes. Ces dernières donnent des sous-gradients de la fonction convexe $-\Theta$, qui seront notés g: avec $z_\ell(\lambda)$ défini comme en (4),

$$g(\lambda) := \sum_{\ell \le \mathcal{L}} g_\ell(\lambda) \quad \text{avec} \quad g_\ell(\lambda) := -h_\ell(z_\ell(\lambda)) . \tag{9}$$

3.1 Méthode des plans sécants

Il est plus simple pour le moment d'ignorer la structure additive de la fonction duale Θ. D'une façon générale, à l'itération courante de l'algorithme dual, le simulateur (2) a été appelé pour un certain nombre de valeurs des variables duales λ^k. Supposons qu'on ait mémorisé **nbun** réponses correspondantes: on dispose du *faisceau d'information* (voir la notation (9), $\Theta(\lambda^k)$ et $g(\lambda^k)$ sont notés respectivement Θ^k et g^k)

$$\{\Theta^1, g^1\}, \ldots, \{\Theta^{\mathbf{nbun}}, g^{\mathbf{nbun}}\} . \tag{10}$$

Pour fixer les idées, on pourra supposer que **nbun** représente le nombre total d'appels au simulateur depuis le début des itérations duales, ce qui est "presque" vrai. Alors, la relation de sous-gradient nous dit que la *fonction des plans sécants*

$$\hat{\Theta}(\lambda) := \min_{k \le \mathbf{nbun}} [\Theta^k + \langle g^k, \lambda - \lambda^k \rangle] \tag{11}$$

est une approximation par excès de la vraie fonction duale: $\hat{\Theta} \ge \Theta$, avec égalité pour chaque point de référence λ^k.

Puisqu'il faut en définitive maximiser Θ, l'idée immédiate et naturelle est alors de maximiser le *modèle* $\hat{\Theta}$. C'est ainsi que l'itération courante de la méthode des plans sécants calcule le nouvel itéré

$$\lambda^+ \in \operatorname{Argmax} \hat{\Theta}(\lambda) . \tag{12}$$

Noter que, $\hat{\Theta}$ étant linéaire par morceaux, ceci est en fait un programme linéaire avec $n+1$ variables et **nbun** contraintes:

$$\begin{cases} \max_{r,\lambda} r , \\ r \le \Theta^k + \langle g^k, \lambda - \lambda^k \rangle \quad k \le \mathbf{nbun} . \end{cases}$$

Dans ce mécanisme, les informations $\{\Theta^k, g^k\}$ sont mémorisées l'une après l'autre dans le faisceau, et **nbun** est effectivement le nombre d'itérations: le faisceau contient *toutes* les informations provenant du simulateur depuis l'itération initiale.

Cette méthode souffre d'un certain nombre d'inconvénients, dont le plus important est qu'elle est intolérablement lente. De fait, elle consiste à parier que le modèle $\hat{\Theta}$ est précis, mais ce pari est grossièrement optimiste. La raison intrinsèque tient à un comportement oscillatoire qui rend la méthode *instable*: en gros, plus un itéré est bon, plus le suivant sera mauvais.

3.2 Méthode de faisceaux

Les méthodes dites de faisceaux répondent à un souci de stabilisation des plans sécants. Entre autres effets bénéfiques, cette stabilisation permet une *compression* du faisceau d'information, de façon à maintenir sa taille limitée tout au long des itérations. Notre technique de compression est expliquée à la fin de cette §3.2.

Reprenant l'algorithme précédent, supposons qu'en plus du faisceau, on dispose à l'itération courante d'un *centre de stabilité* $\hat{\lambda}$. Pour fixer les idées, disons que c'est le λ^k donnant la meilleure valeur duale, ce qui est "presque" vrai; en tous cas c'est l'un des λ^k figurant dans le faisceau. La stabilisation consiste à pénaliser par un terme quadratique la distance au centre de stabilité. Autrement dit, (12) est remplacé par

$$\lambda^+ = \operatorname{argmax}\left[\check{\Theta}(\lambda) - \frac{1}{2t}\|\hat{\lambda} - \lambda\|^2\right],$$

où $t > 0$ module la tension du stabilisateur. Ce programme quadratique a une solution unique, contrairement à (12).

Dans $\mathrm{R}^n \times \mathrm{R}$ (l'espace du graphe de Θ et de $\hat{\Theta}$), il est commode de placer l'origine en $(\hat{\lambda}, \Theta(\hat{\lambda}))$; on pose donc $d = \lambda - \hat{\lambda}$, $\hat{\theta} = \hat{\Theta}(\lambda) - \Theta(\hat{\lambda})$. La fonction des plans sécants (11) s'écrit alors sous la forme

$$\hat{\Theta}(\hat{\lambda} + d) = \Theta(\hat{\lambda}) + \min_{k \le \mathrm{nbun}} [e^k + \langle g^k, d\rangle],$$

où nous avons introduit la notation

$$e^k := \Theta^k + \langle g^k, \hat{\lambda} - \lambda^k\rangle - \Theta(\hat{\lambda}) \ge 0\,;$$

c'est l'erreur faite au centre de stabilité lorsque la fonction concave Θ est remplacée par sa linéarisation en λ^k; plutôt que (10), le faisceau est alors constitué des (e^k, g^k). Avec ces nouvelles notations, notre programme quadratique s'écrit

$$\begin{cases} \max_{\hat{\theta},d}\left[\hat{\theta} - \frac{1}{2t}\|d\|^2\right], \\ \hat{\theta} \le e^k + \langle g^k, d\rangle \qquad k \le \mathrm{nbun}\,. \end{cases} \tag{13}$$

Remarque Le dual de (13) est

$$\begin{cases} \min_\alpha\left[\sum_{k\le\mathrm{nbun}} \alpha^k e^k + \frac{t}{2}\left\|\sum_{k\le\mathrm{nbun}} \alpha^k g^k\right\|^2\right], \\ \alpha^k \ge 0,\ k \le \mathrm{nbun} \qquad \text{et} \quad \sum_{k\le\mathrm{nbun}} \alpha^k = 1\,. \end{cases} \tag{14}$$

Soit $\hat{\alpha}$ une solution de (14). Alors, la solution de (13) s'exprime sous la forme

$$\lambda^+ = \hat{\lambda} + t\hat{G}\,, \quad \hat{\theta}^+ = \hat{\varepsilon} + t\|\hat{G}\|^2\,,$$

où

$$\hat{G} := \sum_{k\le\mathrm{nbun}} \hat{\alpha}^k g^k\,, \quad \hat{\varepsilon} := \sum_{k\le\mathrm{nbun}} \hat{\alpha}^k e^k \ge 0\,. \tag{15}$$

- C'est ce qui explique notre notation pour la tension du stabilisateur: t est un *pas* dans la *direction* $\hat{G}$, laquelle est un "gradient régularisé" de Θ en $\hat{\lambda}$.
- Cela explique aussi que la résolution de (14) passe par le calcul de la matrice de Gram $\langle g^k, g^{k'}\rangle_{k,k'\le\mathrm{nbun}}$; une remarque qui aura son importance en §3.3 pour expliquer les temps de calcul.
- Par combinaison convexe des inégalités de sous-gradient

$$\Theta(\lambda) \le \Theta^k + \langle g^k, \lambda - \lambda^k\rangle = \Theta(\hat{\lambda}) + e^k + \langle g^k, \lambda - \hat{\lambda}\rangle\,,$$

on voit que

$$\Theta(\lambda) \le \hat{\Theta}(\lambda) \le \Theta(\hat{\lambda}) + \hat{\varepsilon} + \langle \hat{G}, \lambda - \hat{\lambda}\rangle \quad \text{pour tout } \lambda \in \mathrm{R}^n. \tag{16}$$

Ceci permet d'écrire une condition d'optimalité approchée et un test d'arrêt: $\hat{\lambda}$ est "bon" quand $\hat{\varepsilon}$ et $\hat{G}$ sont tous deux "petits" (noter qu'ils sont homogènes respectivement à un coût f et des valeurs de contraintes h).

- Un e^k franchement négatif indique une incohérence entre les valeurs de Θ et de g; cette remarque est utile pour diagnostiquer des imperfections dans le simulateur. □

Nous renvoyons au rapport de fin de contrat CERD R31/1J3669/ER238 (reproduit dans [?]) pour une description du code correspondant, lequel suit dans son principe la méthode développée dans [?].

Un point algorithmique, passé sous silence dans [?] et à peine ébauché dans [?], concerne la compression du faisceau (10). D'après (16), la fonction affine $\Theta(\hat{\lambda}) + \hat{\varepsilon} + \langle \hat{G}, \lambda - \hat{\lambda} \rangle$ est une approximation par excès de la vraie fonction duale, tout comme les autres fonctions affines définies par (11). On pourrait ajouter au faisceau (10) la pièce $\{\Theta(\hat{\lambda}) + \hat{\varepsilon}, \hat{G}\}$, cela ne changerait pas $\hat{\Theta}$. Or, la théorie dit que les propriétés de convergence sont conservées si le faisceau à l'itération $k+1$ comporte au moins deux pièces: la pièce "agrégée" ci-dessus, et la nouvelle pièce définie par $\{\Theta^+, g^+\}$. Cela permet de comprimer le faisceau lorsqu'il devient trop encombrant:

Gestion du faisceau La liste d'appel de l'optimiseur comporte un paramètre **memax**: le nombre maximal de pièces dans le faisceau. Notons **knew** l'emplacement où la nouvelle pièce $\{\Theta^+, g^+\}$ va prendre place.

- Tant que **nbun** est strictement inférieur à sa valeur maximale **memax**, on ajoute simplement la nouvelle pièce. Dans cette phase, **knew** = **nbun** + 1, **nbun** s'incrémente de 1 à chaque itération.
- Quand **nbun** = **memax**, on cherche une pièce inactive, i.e. un k tel que (14) ait $\hat{\alpha}^k = 0$. Si une telle pièce est trouvée, on l'appelle **knew**, elle est éliminée et remplacée par la nouvelle. Dans cette phase, **knew** $\leq$ **nbun** et **nbun** reste constamment égal à **memax**.
- Si aucune pièce inactive n'est trouvée, il y a compression totale: le faisceau est complètement nettoyé, pour ne conserver que la pièce agrégée et la nouvelle pièce. A l'issue de cette opération, **nbun** = 2. □

Il convient de mentionner que, si la théorie permet cette stratégie, l'expérience montre que la convergence est considérablement ralentie lorsque beaucoup de pièces sont ainsi rejetées. En particulier, les compressions totales peuvent avoir un effet désastreux. En fait, des stratégies plus raffinées sont possibles, éliminant en particulier le recours à toute compression totale; mais cela impliquerait un substantiel effort informatique.

3.3 Désagrégation du faisceau

Les méthodes précédentes maximisent une fonction concave Θ générale, à l'aide d'un simulateur calculant la fonction et un sous-gradient g. La variante décrite dans cette section est motivée par la structure additive du problème à résoudre.

3.3.1 Principe général

Rappelons d'après (2) que $\Theta = \sum_\ell \Theta_\ell$; de plus, pour un $\lambda = \lambda^k$ donné, le simulateur calcule les valeurs duales locales $\Theta_\ell(\lambda)$ et les sous-gradients locaux $g_\ell(\lambda) = -h_\ell(z_\ell(\lambda))$. Après l'envoi de **nbun** vecteurs-prix λ^k au simulateur, l'optimiseur dispose donc du *faisceau désagrégé*

$$\left\{ \begin{array}{l} \{\Theta_1^1, g_1^1\}, \ldots, \{\Theta_1^{\mathbf{nbun}}, g_1^{\mathbf{nbun}}\}, \\ \quad \vdots \\ \{\Theta_\ell^1, g_\ell^1\}, \ldots, \{\Theta_\ell^{\mathbf{nbun}}, g_\ell^{\mathbf{nbun}}\}, \\ \quad \vdots \\ \{\Theta_\mathcal{L}^1, g_\mathcal{L}^1\}, \ldots, \{\Theta_\mathcal{L}^{\mathbf{nbun}}, g_\mathcal{L}^{\mathbf{nbun}}\}, \end{array} \right. \tag{17}$$

où nous avons posé $\Theta_\ell^k := \Theta_\ell(\lambda^k)$ et $g_\ell^k := g_\ell(\lambda^k)$, tout comme en (10); le faisceau (10) s'obtient par sommation des colonnes de (17).

L'idée de la désagrégation est alors très simple. Aux sections 3.1 et 3.2, l'approximation polyédrique $\hat{\Theta}$ de (11) était utilisée pour calculer le nouvel itéré λ^+. Or, dans le cas décomposable, chaque fonction duale locale Θ_ℓ de (2) a aussi sa propre approximation polyédrique:

$$\Theta_\ell(\lambda) \leq \hat{\Theta}_\ell(\lambda) := \min_{k \leq \mathbf{nbun}} [\Theta_\ell^k + \langle g_\ell^k, \lambda - \lambda^k \rangle] . \tag{18}$$

Par sommation sur les agents locaux, on en déduit $\Theta \leq \sum_\ell \Theta_\ell$. D'après (18), la somme des $\hat{\Theta}_\ell$ est une somme de min; elle est inférieure à $\hat{\Theta}$ qui, d'après (11), est le min d'une somme:

$$\Theta \leq \sum_{\ell \leq \mathcal{L}} \hat{\Theta}_\ell \leq \hat{\Theta} . \tag{19}$$

Autrement dit, la désagrégation fournit un modèle de la fonction duale plus précis que le schéma classique (11). Cette désagrégation peut être greffée aussi bien sur la §3.1 que sur la §3.2. Dans la méthode de plans sécants pure, par exemple, (12) sera remplacé par

$$\lambda^+ \in \operatorname*{Argmax}_{\lambda} \sum_{\ell \le \mathcal{L}} \Theta(\lambda) .$$

Nous ne nous intéressons ici qu'à la version stabilisée de la §3.2, où le nouvel itéré est maintenant

$$\lambda^+ = \operatorname*{argmax}_{\lambda \in \mathbb{R}^n} \Big[\textstyle\sum_{\ell \le \mathcal{L}} \Theta_\ell(\lambda) - \frac{1}{2t} \|\lambda - \hat{\lambda}\|^2 \Big] . \tag{20}$$

L'algorithme résultant se présente schématiquement comme suit (dans la description qui suit, le test d'arrêt issu de (16) s'effectue après chaque résolution du problème quadratique (20) à l'Étape 1).

Méthode de faisceaux désagrégée On dispose du faisceau désagrégé (17) et du centre de stabilité $\hat{\lambda}$.

ETAPE 1 (Préparation de l'itération) Former la matrice de Gram et calculer un premier candidat par résolution de (20) – en fait le dual (22) ci-dessous.

ETAPE 2 (Recherche curviligne) Ajuster t par une suite de résolutions de (22). Obtenir ainsi un nouvel itéré λ^+ qui donnera
- soit un pas nul: on se contente d'enrichir le faisceau,
- soit un pas de descente: $\hat{\lambda}$ est déplacé en λ^+.

ETAPE 3 (Gestion du faisceau) Déterminer dans le faisceau $\mathcal{L}$ emplacements $\{\texttt{knew}_\ell\}_{\ell \le \mathcal{L}}$ où mettre les $\mathcal{L}$ nouveaux éléments $\{\Theta_\ell^+, g_\ell^+\}_{\ell \le \mathcal{L}}$. Pour ce faire, effacer si besoin est $\mathcal{L}$ éléments, ou même faire une compression totale (voir la gestion du faisceau en §3.2). □

Avec $\mathcal{L} = 1$, cet algorithme n'est autre que la méthode de faisceaux "standard". Puisque $\mathcal{L}$ détermine le nombre de termes dans le problème (19), plus $\mathcal{L}$ est grand, plus le modèle est ajusté, et plus la convergence devrait être rapide. Toutefois, de grandes valeurs de $\mathcal{L}$ soulèvent des difficultés que nous présentons maintenant.

3.3.2 Difficultés nouvelles

Avec ou sans terme quadratique pour calculer le nouvel itéré λ^+, le but de la désagrégation est d'obtenir une meilleure approximation de Θ, donc un meilleur λ^+, et donc en définitive de réduire le nombre d'itérations de l'algorithme non différentiable. Cependant, cette accélération a son prix, et ceci sur deux plans.

Encombrement mémoire Dans (10) comme dans (17), chaque accolade contient un nombre (une valeur duale) et un n-vecteur (un sous-gradient). Après nbun itérations, le faisceau agrégé (10) nécessitait $(n+1) \times \mathbf{nbun}$ mémoires. Ici, le faisceau désagrégé est $\mathcal{L}$ fois plus encombrant, avec $\mathcal{L} = 10^2$ (voir §2.2). A priori, il nécessite $(n+1) \times \mathbf{nbun} \times \mathcal{L}$ mémoires; tenir compte du creux éventuel du problème peut devenir crucial (par exemple pour le problème X).

Par ailleurs, pour mettre (20) sous une forme analogue à (13), on attribue à chaque agent local ℓ sa propre "variable verticale" $\hat{\theta}_\ell$ et ses nbun erreurs de linéarisation $(e_\ell^k)^{k \le \mathbf{nbun}}$. Le problème quadratique a maintenant $n + \mathcal{L}$ variables et $\mathbf{nbun} \times \mathcal{L}$ contraintes, au lieu de $n+1$ et nbun respectivement pour (13):

$$\begin{cases} \max_{\hat{\theta}_\ell, d} \big[\sum_\ell \hat{\theta}_\ell - \frac{1}{2t} \|d\|^2 \big] , \\ \hat{\theta}_\ell \le e_\ell^k + \langle g_\ell^k, d \rangle , \quad k \le \mathbf{nbun} , \qquad \text{pour chaque } \ell \le \mathcal{L}. \end{cases} \tag{21}$$

Comme en §3.2, on résout en fait le dual, qui s'écrit

$$\begin{cases} \min_\alpha \Big[\sum_{k,\ell} \alpha_\ell^k e_\ell^k + \frac{t}{2} \Big\| \sum_{k,\ell} \alpha_\ell^k g_\ell^k \Big\|^2 \Big] , \\ \alpha_\ell^k \ge 0, \ k \le \mathbf{nbun} \quad \text{et} \ \sum_{k \le \mathbf{nbun}} \alpha_\ell^k = 1 , \qquad \text{pour chaque } \ell \le \mathcal{L}. \end{cases} \tag{22}$$

La matrice de Gram associée à ce problème est maintenant de la forme $\langle g_\ell^k, g_{\ell'}^{k'} \rangle_{k,k' \le \mathbf{nbun}}^{\ell,\ell' \le \mathcal{L}}$. Elle nécessite de l'ordre de $(\mathbf{nbun} \times \mathcal{L})^2$ némoires (la moitié, du fait de sa symétrie), soit $\mathcal{L}^2$ fois plus qu'en version non désagrégée. Plus grave: cette matrice pourrait bien être pleine, même si le problème est creux. Résultat: une décomposition du problème en beaucoup d'agents locaux ne permettra de conserver que peu de sous-gradients dans le faisceau, au prix d'une dégradation éventuelle des performances.

Temps de calcul Il a été vu dans [?] qu'en situation agrégée ($\mathcal{L} = 1$), le temps passé dans l'optimiseur était faible, comparé au simulateur. Sur un problème réel, ce temps était négligeable pour une décomposition N, et de l'ordre de 20% pour une décomposition X. Il n'en est pas nécessairement de même ici.

L'analyse de l'optimiseur montre que son temps d'exécution se divise en deux parties: résolution de (22) et calcul de la matrice de Gram; le reste est négligeable. Notons déjà ici que ce temps est peu influencé par la valeur de n; plus précisément, n n'intervient que dans les produits scalaires entre sous-gradients.

- L'algorithme quadratique résolvant (22) requiert un nombre d'itérations imprévisible, mais moins que linéaire par rapport au nombre $\mathtt{nbun} \times \mathcal{L}$ de variables α_ℓ^k. De fait, chaque exécution de cet algorithme ne diffère de la précédente que par des modifications des données affectant $\mathcal{L}$ variables seulement: il s'agit plutôt d'une ré-optimisation. Dans ces conditions, l'impact d'une valeur $\mathcal{L} = 10^2$ sur le programme quadratique est difficile à évaluer. Si nbun est grand (disons $\gg 10$) la ré-optimisation n'affecte qu'une petite partie des variables (disons $\ll 10\%$). Si nbun est petit, les contraintes simpliciales sur les α_ℓ^k rendent le problème "facile" (si $\mathtt{nbun} = 1$, (22) n'a qu'un point réalisable).
- Quant à la matrice de Gram, elle n'est évidemment pas recalculée à chaque itération. En fait, chaque itération ajoute ou insère $\mathcal{L}$ éléments dans le faisceau; pour mettre à jour cette matrice, on calcule

$$\langle g_\ell^k, g_{\ell'}^+ \rangle \quad \text{pour } k \leq \mathtt{nbun} \text{ et } \ell, \ell' \leq \mathcal{L},$$

soit de l'ordre de $\mathtt{nbun} \times \mathcal{L}^2$ produits scalaires par itération. Pour $\mathcal{L} = 10^2$, cette phase de calcul est donc $\mathcal{L}^2 = 10^4$ fois plus chère qu'en situation agrégée (si l'on ne tient pas compte du creux éventuel).

3.3.3 Les remèdes

La section précédente a montré que la désagrégation est une opération coûteuse, qui risque de submerger l'optimiseur sous une avalanche de données; et ceci surtout dans le problème X. Comme il a été vu en §2.2.2, chaque appel au simulateur produit $\mathcal{L} = 10^2$ sous-gradients, ayant chacun $\mathcal{I} \times T = 10^4$ coordonnées; soit un total de 10^6 données. Il s'agit de les charger dans le faisceau, d'en faire les produits scalaires, et d'envoyer le tout au solveur quadratique.

Une contrepartie de l'inconvénient ci-dessus est la possibilité d'un creux structurel dans le faisceau. Il se peut en effet que chaque fonction Θ_ℓ ne dépende pas de *toutes* les variables duales. Ceci est un privilège de la désagrégation: s'il n'y a qu'une fonction duale Θ, elle dépend certainement de toutes les variable duales (sinon, il y aurait des variables duales inutiles, quelque chose n'irait pas dans la formulation!) Bien que les avantages apportés par le creux ne soient pas à démontrer, quelques commentaires sont nécessaires, en relation avec la §3.3.2 ci-dessus.

(i) Chaque colonne du faisceau (17) a non plus $\mathcal{L} \times n$ éléments mais $\sum n_\ell$, où n_ℓ est le nombre de variables dont dépend l'unité ℓ. Si, comme c'est le cas dans le problème X, chaque n_ℓ est de l'ordre de $10^{-2}n$, on a gagné un facteur 100 sur l'encombrement mémoire.

Toutefois, cela a un impact sur la gestion du faisceau (cf. Etape 3 dans l'algorithme §3.2). Supposons $\mathtt{nbun} = \mathtt{memax}$. Chaque g_ℓ^+ occupe n_ℓ mémoires, et doit prendre la place d'un autre sous-gradient, disons $g_{\ell'}^{\mathtt{knew}_\ell}$. Or, ce dernier occupe $n_{\ell'}$ mémoires, et le remplacement implique un gros travail informatique si $n_\ell \neq n_{\ell'}$. C'est pourquoi nous avons choisi d'imposer $\ell' = \ell$ dans ce remplacement: un nouvel élément dans la colonne ℓ ne peut remplacer un élement que dans la même colonne ℓ. Si (22) ne produit aucun $\hat{\alpha}_\ell^k$ nul, il y a donc compression totale (voir la gestion du faisceau en §3.2).

(ii) Les quelque $\mathtt{nbun} \times \mathcal{L}^2$ produits scalaires nécessaires à la mise à jour de la matrice de Gram sont maintenant creux, au lieu de consommer chacun n opérations.

De plus: si le problème est très creux, un bon nombre de paires ℓ, ℓ' sont telles que les agents ℓ et ℓ' n'ont aucune variable duale en commun (c'est le cas pour le problème X, nettement structuré en blocs distincts). Les produits scalaires correspondants $\langle g_\ell^k, g_{\ell'}^{k'} \rangle$ sont nuls, pour tout k et k'; d'où matrice de Gram creuse. Néanmoins, ici encore, profiter de ce creux impliquerait un gros travail informatique: le solveur quadratique tient pour acquis que la matrice de Gram se présente comme une matrice symétrique d'ordre $\mathtt{nbun} \times \mathcal{L}$. Noter que ce n'est pas le *calcul* de ces produits scalaires "vides" qui est coûteux, mais leur *encombrement*.

(iii) L'un des effets bénéfiques de la méthode de faisceaux standard est que chaque pièce $\{\Theta^+, g^+\}$ apporte une information "utile", de sorte que la nouvelle approximation des plans sécants $\hat{\Theta}^+$ diffère sensiblement de l'ancienne[2]. Toutefois, cette propriété n'est pas transmise au niveau désagrégé: on peut avoir $\hat{\Theta}_\ell^+ \equiv \hat{\Theta}_\ell$

2. Cela n'est vrai à coup sûr qu'en cas de pas nul; après un pas de descente, il peut arriver exceptionnellement que $\hat{\Theta}^+ \equiv \hat{\Theta}$.

pour certains ℓ sans l'avoir pour leur somme. Il est clair que, si $\hat{\Theta}_\ell^+ \equiv \hat{\Theta}_\ell$, la nouvelle pièce ℓ est inutile et peut avantageusement être jetée.

Pour mettre en oeuvre cette idée, nous avons choisi de filtrer les pièces suivant la valeur de $\Theta_\ell(\lambda^+) - \hat{\Theta}_\ell(\lambda^+)$. Sachant que ces nombres sont positifs, on les compare à leur moyenne: la pièce ℓ est décrétée inutile si

$$\Theta_\ell(\lambda^+) - \hat{\Theta}_\ell(\lambda^+) \leq \texttt{armuse} \frac{\Theta(\lambda^+) - \hat{\Theta}(\lambda^+)}{\mathcal{L}} . \tag{23}$$

Ici **armuse** est un coefficient de type Armijo, initialisé à une valeur comprise entre 0 et 1, et diminuant chaque fois que l'effet du filtrage paraît néfaste.

Cette technique peut avoir une influence négative sur la vitesse de convergence de l'optimiseur (puisqu'elle risque de gâter la qualité du modèle $\hat{\Theta}$). En contrepartie, elle permet d'économiser du temps, aussi bien dans le calcul des produits scalaires que dans la résolution du programme quadratique (22). Elle permet aussi d'économiser de la mémoire mais, pour les mêmes raisons qu'en (ii) ci-dessus, nous n'avons pas profité de cet aspect: cela aurait impliqué un investissement informatique trop important.

(iv) C'est incontestablement sur la résolution du programme quadratique (22) que repose la plus lourde charge de l'optimiseur, sinon en temps d'exécution, du moins en complexité; tout gravite autour d'elle. Cette résolution se fait par gestion de contraintes actives: à chaque itération,

. une partie des α sont fixés à 0,

. les autres sont donnés par les conditions d'optimalité du problème réduit associé à (22); c'est un système linéaire dont la matrice est symétrique définie positive.

L'avantage de cette méthode est qu'elle est bien adaptée aux ré-optimisations d'une exécution sur l'autre. L'algorithme quadratique est alors schématiquement le suivant.

Solveur quadratique On part d'une liste A d'indices (k,ℓ) actifs.

ETAPE 1 (Calcul d'un candidat) Résoudre par Choleski le système donnant $(\hat{\alpha}_\ell^k)_{(k,\ell)\notin A}$.

ETAPE 2 (Test de positivité) Si le candidat ainsi obtenu a une composante négative, l'ajouter dans A. Mettre à jour la matrice de Choleski et boucler en 1.

ETAPE 3 (Test d'arrêt) Tester le signe des multiplicateurs associés aux contraintes $\alpha_\ell^k \geq 0$, $(k,\ell) \in A$. Si l'un d'eux est négatif, retrancher son indice de A. Mettre à jour la matrice de Choleski et boucler en 1. □

Rappelons que le solveur quadratique utilisé est le code **q2edf**, developpé par K.C. Kiwiel (Interfaces Institute, Varsovie). En version originale, il utilise le faisceau sous forme d'une matrice dont les dimensions sont **nbun** $\times \mathcal{L}$ et n. Une version 2 a donc été écrite à l'Inria, acceptant un faisceau creux.

Par ailleurs, considérons comme en (15) le candidat obtenu à l'Etape 1: $\hat{\varepsilon} = \sum \hat{\alpha}_\ell^k e_\ell^k$, $\hat{G} = \sum \hat{\alpha}_\ell^k g_\ell^k$. Le test d'arrêt consiste à vérifier s'il forme un point admissible dans (21), c'est à dire si

$$\hat{\varepsilon} + t\|G\|^2 \leq e_\ell^k + t\langle g_\ell^k, G\rangle \quad \text{pour } k = 1,\ldots,\texttt{nbun} \text{ et } \ell \leq \mathcal{L}.$$

Ceci coûte cher: un produit scalaire pour chacun des **nbun** $\times \mathcal{L}$ éléments du faisceau. Dans **q2edf** (versions 1 et 2), les membres de droite sont calculés à partir de la matrice de Gram, utilisant l'expression de $\hat{G}$: on a

$$\langle g_\ell^k, \hat{G}\rangle = \sum_{(k',\ell')\notin A} \hat{\alpha}_{\ell'}^{k'} \langle g_\ell^k, g_{\ell'}^{k'}\rangle .$$

Cet artifice peut ne pas être payant dans le cas creux puisque le calcul direct de $\langle g_\ell^k, \hat{G}\rangle$ est alors économique – bien que $\hat{G}$ soit plein; qui plus est, la matrice de Gram n'exploite pas le creux (voir (ii) ci-dessus). Une version 3 a donc été écrite à l'Inria, où le test d'arrêt se fait directement sur le faisceau, à l'aide d'un sous-programme calculant le produit scalaire d'un vecteur creux (g_ℓ^k) avec un vecteur plein $(\hat{G})$.

Conclusion La désagrégation dans une méthode de faisceaux est une technique simple dans son principe. Du point de vue théorique, il n'est pas difficile de voir qu'elle jouit des mêmes propriétés de convergence que la version standard. Toutefois, une implémentation efficace ne va pas sans difficultés d'ordre informatique, en cas de surabondance d'informations provenant d'un simulateur désagrégé. Dans la résolution de ces difficultés, nous avons surtout cherché l'efficacité en temps de calcul; quant à l'encombrement mémoire, nous nous sommes contentés de compromis compatibles avec un investissement informatique raisonnable.

4 Description des routines

D'un point de vue informatique, le code de faisceaux désagrégés fonctionne schématiquement comme suit.

Routine sprom La routine-chapeau **sprom** commence par quelques vérifications et initialisations; elle appelle ensuite la sous-routine **transpar**, qui définit la structure creuse par ligne (utilisée par **q2edf**) à partir de la structure creuse par colonne donnée par l'utilisateur (cf. **icr, ncr**). Elle appelle ensuite **sproma**, qui constitue le corps de la méthode.

Voir la liste-fortran pour la description des paramètres de **sprom**. Par ailleurs, un certain nombre de paramètres supplémentaires figurent dans un bloc commun:

common/cospro/armuse,armul,dfrel,stopqp,itmqp,mxsprb,lgram,lworqp

De ces paramètres, seuls les trois premiers peuvent être éventuellement modifiés par un utilisateur averti (les suivants concernent **q2edf** et devraient rester tels quels).

Paramètre armuse Initialisation du coefficient apparaissant dans (23); **armuse**=1. dans la version standard.

Paramètre armul Coefficient multiplicateur: **armuse** est multiplié par **armul** chaque fois que la stratégie de filtrage paraît néfaste; **armul**=0.5 dans la version standard.

Paramètre dfrel Coefficient permettant de diagnostiquer des progrès insuffisant de l'algorithme: si la décroissance prédite pour Θ est inférieure à **dfrel** en valeur relative, alors t est augmenté et **armuse** est multiplié par **armul**. Cette heuristique est destinée à lutter contre le phénomène de convergence interminable. Si ce phénomène se produit, on peut tenter d'augmenter **dfrel**, qui vaut 10^{-5} dans la version standard.

Sous-routine sproma Après quelques initialisations, la sous-routine **sproma**

- met à jour la matrice de Gram par un appel à **spromc**,
- effectue un premier appel de **q2edf**, via l'interface **spromf**, de façon à obtenir un premier candidat λ^+ (rappelons que chaque appel à **q2edf** est suivi d'un test d'arrêt, lequel est effectué dans **sprome**),
- appelle la sous-routine de recherche curviligne **spromb** pour l'ajustement du pas t et de λ^+,
- filtre les pièces inutiles et gère le coefficient de filtrage **armuse** mentionné en §3.3.3(iii) (voir ci-dessus le paramètre **armul** dans le bloc commun **cospro**),
- vérifie s'il y a de la mémoire pour insérer les nouveaux éléments dans le faisceau (sous-routine **spromg**, voir ci-dessous),
- en cas de descente obtenue par **spromb**, met à jour dans **spromd** le coefficient de pénalisation t dans le terme quadratique de (21).

Ceci constitue une itération.

Sous-routine spromb Ajuste le pas t par appels successifs au solveur quadratique **q2edf**. Chacun de ces appels est suivi d'un appel au simulateur pour calcul de fonction-gradient au nouveau candidat. La recherche curviligne utilisée est décrite dans [?].

Sous-routine spromc Met à jour la matrice de Gram de (22), calculant les produits scalaires $\langle g_\ell^k, g_{\ell'}^+ \rangle$, pour $k \leq$ **nbun** et $\ell, \ell' \leq \mathcal{L}$ filtrés par **sproma**. Ces produits scalaires sont calculés par la sous-routine **pscreux**, à laquelle sont transmis les vecteurs **icr, ncr** caractérisant la structure creuse.

Sous-routine spromd Met à jour la norme du terme quadratique de (21), suivant la formule dite de *quasi-Newton du pauvre* dans [?].

Sous-routine sprome Effectue le test d'arrêt issu de (16). Ce test d'arrêt vérifie si

$$\hat{e} \leq \texttt{epsrel} \times |\Theta(\hat{\lambda})| \quad \text{et} \quad \|\hat{G}\| \leq \texttt{eta},$$

où **epsrel** et **eta** sont donnés par l'utilisateur. Dans l'affirmative, **sprom** stoppe avec **mode**=1 (sortie normale).

Sous-routine spromf Interface avec le solveur quadratique `q2edf`; le mode d'entrée dans `q2edf` (paramètre `modeqp`) dépend des modifications apportées au faisceau depuis le dernier appel: changement de t, pas nul, pas de descente, compression totale,... Par ailleurs, `spromf` gère les conditions de retour de `q2edf` (convergence ou fin anormale) et calcule la solution primale λ^+ de (20) à partir de la solution duale $\hat{\alpha}$ de (22).

Sous-routine spromg Pour chaque g_ℓ^+ filtré par `sproma`, cherche un indice k ayant $\hat{\alpha}_\ell^k = 0$. En cas de succès, pose $\texttt{knew}_\ell$ égal au k ayant le plus grand e_ℓ^k. En cas d'échec, appel de la sous-routine `totale`, qui ne conserve qu'un élément pour chaque unité locale: le sous-gradient régularisé G_ℓ.

5 Résultats numériques

Nous comparons maintenant la méthode de faisceaux en versions standard et désagrégée, sur les trois problèmes présentés en §2.2.

5.1 Problème N

Sur les maquettes de la §2.2.1, la méthode de faisceaux trouve toujours l'optimum dual *exact*: au pire, on obtient dans le test d'arrêt selon (16) les ordres de grandeur $\hat{\epsilon} \simeq 10^{-7}$ et $|G| \simeq 10^{-10}$, pour des valeurs $\Theta \simeq 10^5$ et $\lambda \simeq 0.1$.

Rappelons que le problème dual a ici $n = T = 48$ variables. Nous avons testé 7 exemples; les 5 premiers ont $\mathcal{L} = \mathcal{I} = 100$ unités; les deux derniers en ont 10, avec pour le septième un pic de demande saturant la capacité de production; cela donne des variables duales pouvant atteindre la valeur 150 (comparée à 0.1). Pour chaque exemple, nous avons utilisé systématiquement deux intialisations des variables duales: $\lambda^1 = 100$, ce qui n'a aucun rapport avec la réalité, et λ^1 proche de l'optimum, via l'heuristique habituelle d'empilement des groupes. En version standard, la méthode de faisceaux était autorisée à mémoriser $\texttt{memax} = 100$ sous-gradients (ce qui était amplement suffisant vu le nombre d'itérations). En version désagrégée, ce nombre était $10 \times \mathcal{L}$ sous-gradients; cela provoquait des compressions dès la 11e itération, mais pas de compression totale.

Sur chacun de ces exemples, nous avons comparé:
- le nombre total d'appels au simulateur,
- le temps de calcul (en secondes sur un Sparc20),
- le pourcentage des temps consommés respectivement par: la gestion de matrice de Gram, le solveur quadratique et les simulations.

Les résultats sont consignés dans la Table 1. Chaque bloc de cette table comporte 4 nombres, qui diversifient l'initialisation λ^1 et la méthode, et qui se lisent comme suit:

Standard, $\lambda^1 = 100$	Standard, λ^1 ajusté
Désagrégé, $\lambda^1 = 100$	Désagrégé, λ^1 ajusté

Par exemple, la méthode de faisceaux standard nécessite sur le premier problème 70 [resp. 41] simulations pour une initialisation éloignée [resp. ajustée]; pour la version désagrégée, ces nombres passent à 71 et 18. Concernant les pourcentages, noter qu'ils ne se somment pas à 100 mais plutôt à environ 80. Les 20% restants se distribuent entre les autres phases du programme et l'instrument de mesure lui-même: les mêmes exécutions sans mesurer les temps sont 80-90% plus courtes.

Pour le 7e problème les résultats de la version désagrégée avec initialisation à 100 sont marqués d'un *. Dans ce test, une compression totale a été effectuée; elle n'a d'ailleurs eu guère d'influence sur le résultat final: en portant `memax` de $10 \times \mathcal{L}$ à $12 \times \mathcal{L}$, il n'y a plus de compression totale et le nombre de simulations passe de 150 à 147.

A propos de cette table, mentionnons tout d'abord que la convergence s'effectue beaucoup mieux, y compris pour la version standard, que dans le code `proxmyqn` testé dans [?], [?]. De fait, nous avons apporté à ce code quelques améliorations supprimant le phenomène de convergence interminable qui apparaissait parfois (voir la colonne "Partiel" dans la table § 4 de [?]). Par ailleurs, le déroulement des itérations ne dépend plus des tolérances d'arrêt demandées par l'utilisateur.

Globalement, les résultats montrent l'effet bénéfique de la désagrégation (réduction sensible du nombre de simulations) ainsi que son prix: sur cet exemple où les simulations ne coûtent rien, le temps CPU consommé par l'optimiseur devient important. Admettant que la convergence en une centaine d'itérations est une situation

	#simul		CPU		% Gram		% pg.quad.		% simul	
Pb.1	70	41	7	4	0	1	1	2	87	79
	71	18	322	52	34	35	45	42	1	2
Pb.2	193	52	23	6	1	0	6	1	74	79
	141	35	784	161	30	39	49	42	1	1
Pb.3	88	40	8	3	0	0	1	0	80	81
	64	19	360	45	35	36	46	40	1	2
Pb.4	200	93	18	49	1	1	5	4	74	79
	118	46	619	170	34	46	46	35	1	2
Pb.5	174	105	17	10	1	1	7	6	78	70
	139	46	603	192	37	47	43	33	1	1
Pb.6	166	168	8	4	5	5	34	27	26	27
	126	99	7	6	38	35	20	18	20	10
Pb.7	146	99	3	3	5	3	30	28	25	39
	*150	93	*8	6	*40	42	*17	20	*13	13

TAB. 1 – *Résultats sur le problème N*

typique, ce temps peut être en gros estimé à une dizaine de minutes[3]. Ceci vaut pour une désagrégation avec $\mathcal{L} = 100$, sachant que $\mathcal{L} = 10$ (Pbs. 6 et 7) nous ramène à des temps très raisonnables, en tous cas négligeables devant un simulateur "grandeur nature".

Dans quelques cas (Pb.1 avec $\lambda^1 = 100$, Pb.7), la désagrégation n'est d'aucune efficacité, et la cause de telles "anomalies" n'est pas entièrement claire[4]. Pour vérifier si cela ne provenait pas du filtrage des pièces inutiles (§3.3.3(iii)), nous avons conduit des expériences testant plusieurs stratégies de filtrage. La table 2 donne les résultats comparatifs sur les mêmes problèmes que la table 1, mais uniquement avec l'initialisation $\lambda^1 = 100$. Chaque bloc de cette table comporte 3 nombres diversifiant les stratégies:

Aucun filtrage: `armuse`=0 dans (23)
Filtrage prudent: `armuse`=1/2, diminué vite
Filtrage agressif: `armuse`=1, diminué lentement

Le filtrage agressif est à peu près celui utilisé dans la table 1. Dans la table 2, le nombre d'itérations est le nombre de mises à jour de la matrice de Gram; le nombre de pièces est le nombre cumulé de pièces ajoutées au faisceau à chaque itération; c'est en gros lui qui conditionne le temps total d'exécution (en cas de non-filtrage – $m = 0$ et supposant aucune erreur d'arrondi dans (23) – ce nombre est #iter $\times$ $\mathcal{L}$). A noter: les temps CPU reflètent mieux la réalité que dans la table 1, car les mesures ont été supprimées. Remarquer l'assez grande variabilité de ces temps; on en conclut que leur valeur est indicative uniquement. Les pourcentages reportés dans la table 1 ne figurent pas ici: ils ne sont pas significatifs.

Pour que le filtrage ait une nette supériorité, il faudrait qu'il apporte une très nette diminution du CPU (pour que ce gain ne soit pas détruit par l'augmentation du nombre de simulations, avec un simulateur grandeur nature). La table 2 ne révèle pas d'évidence que ce soit le cas. Ici encore, une anomalie apparaît sur le problème 2, où c'est le filtrage prudent qui garde en fin de compte le moins de pièces,... ce qui ne l'empêche pas de converger le plus vite (méditer la Note 4, de nouveau).

5.2 Problème X

Des trois problèmes considérés, celui-ci est le plus gros, avec ses $n = 10848$ variables duales et $\mathcal{L} = 188$ unités. C'est aussi le plus significatif puisqu'il est le seul présentant un creux exploitable. En fait c'est ce problème qui a révélé les vraies difficultés de la désagrégation et leurs remèdes.

Le creux du problème est tel que l'ensemble des 188 gradients tiennent en 21696 mémoires, soit 1% de $188 \times 10848 = 10^6$; la gestion du faisceau devient alors une entreprise raisonnable. Comme il a été vu en §3.3.3, cela n'a pas d'impact sur la gestion de la matrice de Gram, mais seulement sur son calcul (produits scalaires

3. Noter que, pour tous les essais de cette série, le programme tenait entièrement en mémoire centrale; il s'agit donc de temps "vrai" (aux erreurs de mesure près), à l'exclusion de tout swap.

4. En optimisation non différentiable, on n'est jamais sûr de rien, puisque les prédictions théoriques sont de toutes façons très pessimistes par rapport aux vitesses observées.

	#iter	#simul	#pièces	CPU
Pb.1	41	62	4040	306
	43	68	3600	319
	53	71	3000	302
Pb.2	64	114	6400	488
	51	94	3730	390
	94	141	4283	641
Pb.3	39	50	3900	350
	35	48	2967	324
	54	64	2470	304
Pb.4	72	102	7200	525
	75	102	5160	521
	87	118	4284	530
Pb.5	65	93	6190	428
	75	111	5640	512
	96	140	4020	536
Pb.6	76	94	760	5
	85	109	628	5
	117	126	598	7
Pb.7	93	113	926	6
	109	139	866	7
	131	157	685	8

TAB. 2 – *Comparaison de divers filtrages*

creux). L'exploitation du creux, tant pour l'encombrement du faisceau que pour le calcul des produits scalaires, a été testée sur le présent problème. *Sans elle, la méthode ne serait en fait pas viable.*

Toutefois, nous n'avons pu tester faute de temps l'autre technique donnée en §3.3.3(iii), à savoir le filtrage des pièces inutiles dans le faisceau: dans le programme testé, chaque appel au simulateur produit systématiquement 188 éléments qui sont tous incorporés au faisceau et fournis au solveur quadratique. Nous n'avons pas non plus testé la version 3 de `q2edf`. Le test d'arrêt est celui de la version 2, qui se fait par balayage de la matrice de Gram, et non par produit scalaire creux. Cela consomme les 3/4 du temps de résolution de (22), soit 35-40% du temps total de résolution du problème dual, suivant la valeur de `memax`.

La table 3 résume les résultats obtenus: méthode standard en première ligne, et désagrégation complète ($\mathcal{L} = 188$) sur les lignes suivantes, en fonction de la taille maximale accordée au faisceau: `memax` $= 5\mathcal{L}, 10\mathcal{L}, 15\mathcal{L}$ respectivement.

On pourra noter que le temps d'une simulation est ici d'environ 2s. Par comparaison, une itération d'optimiseur prend respectivement 10, 20 et 30 secondes suivant les valeurs de `memax`; et c'est environ 20% de ces temps qui sont consommés par les produits scalaires (creux).

`memax`	#simul	CPU	% Gram	% pg.quad.	% simul
250	200	1023	5	23	39
940	112	1904	14	48	10
1880	113	3912	13	54	6
2820	107	6204	11	56	4

TAB. 3 – *Résultats sur le problème X*

Les présents résultats sont à considérer avec la plus extrême prudence. Tout d'abord, ils ont été obtenus sur des versions d'optimiseur maintenant périmées. De plus, certaines incohérences ne sont pas entièrement éclaircies et font penser que le logiciel testé n'était pas exempt de toute erreur.

En fait, notre expérience sur des problèmes combinant d'aussi grandes tailles, tant en nombre de variables qu'en nombre d'agents locaux, est encore trop limitée pour tirer des conclusions définitives. Nous ne nous permettrons qu'une observation: la désagrégation est pénalisée par son programme quadratique plus complexe, et ce handicap est lourd à rattraper. Malgré une convergence plus rapide – déjà obtenue sur le problème N –, on

a du mal à diminuer les temps de calcul, même avec un simulateur grandeur nature (mais sans prise en compte des équations de réseau, toutefois).

5.3 Problème S

Rappelons que

- ce problème se présente formellement comme le problème N, la principale différence portant sur le nombre n de variables duales;
- nous en avons considéré trois exemples où n vaut respectivement 312, 760, 1016;
- sur chacun de ces exemples, nous avons testé la méthode de faisceaux sous trois formes: la version standard ($\mathcal{L} = 1$) et deux versions désagrégées ayant respectivement $\mathcal{L} = 13$ et $\mathcal{L} = 123$.

Les résultats figurent dans la Table 4, qui donne pour chaque problème: la valeur duale obtenue, le nombre total de simulations, puis les temps d'exécution (en secondes). Chaque bloc de cette table contient trois nombres qui diversifient la variante de faisceaux:

Standard (memax = 1000)
Désagrégation 13 (memax = 260 = 20$\mathcal{L}$)
Désagrégation 123 (memax = 1230 = 10$\mathcal{L}$)

C'est seulement ici que la méthode prend sa pleine mesure, montrant sa robustesse et sa rapidité potentielle d'exécution.

	Θ opt.	$\#$ simul	total	faisceaux	simul
312	407260.	500	2417	65	2316
	407288.	93	521	78	436
	407288.	78	1184	810	367
760	407137.	500	5815	143	5588
	407288.	88	1242	222	1005
	407287.	72	2583	1759	811
1016	407148.	500	7658	176	7369
	407287.	97	1825	354	1450
	407288.	92	4675	3296	1359

TAB. 4 – *Résultats sur le problème S*

Sur aucun exemple, la version standard n'a obtenu le test d'arrêt après 500 simulations. Ce test d'arrêt était cependant le même pour toutes les versions: stopper lorsque (16) est satisfait avec $\varepsilon \leq 10^{-3}|\Theta(\hat{\lambda})|$ et $|G| \leq 1$ (ce qui correspond en gros à une violation de $1/\sqrt{n}$ MW pour chaque contrainte). Par contraste, le test d'arrêt a été obtenu par toutes les versions désagrégées, qui ont trouvé un optimum au franc près, en satisfaisant la demande au kW près[5].

Quant à la rapidité d'exécution, on voit également que la consommation de l'optimiseur en temps CPU peut être substantiellement compensée par l'économie en nombre de simulations. Toutefois cette économie peut être tuée par une désagrégation trop brutale: augmenter $\mathcal{L}$ ne doit se faire qu'à bon escient.

6 Conclusions et avenir

La désagrégation est capable d'accélérer substantiellement les méthodes de faisceaux (division du nombre de simulations par des facteurs pouvant dépasser 5). Cela se paie en encombrement-mémoire, mais aussi en temps de calcul consommé par l'optimiseur: si l'on n'y prend pas garde, ce temps peut largement dominer celui du simulateur.

Les situations où la désagrégation apporte un réel gain global en temps de calcul sont assez complexes à analyser. Elles dépendent d'un certain nombre de facteurs, parmi lesquels figurent:
- le creux du problème.

5. Il s'agit bien sûr d'une convexification: c'est le problème dual (3) qui est résolu avec une telle précision, et non le vrai problème (1). Pour une étude du *saut de dualité*, différenciant (1) et (3), voir par exemple [?].

- le nombre de sous-gradients à conserver dans le faisceau,
- le temps de simulation,
- le temps de produit scalaire entre deux sous-gradients locaux,
- et en définitive, le temps nécessaire à `q2edf`, difficile à prévoir.

La décomposition temporelle (problèmes N et S), qui ne présente aucun creux, est d'analyse relativement facile. C'est ce qui a été fait dans cette étude, qui montre clairement les potentialités de la désagrégation: une technique très efficace dans ce type applications. Il reste toutefois à mieux cerner les impacts respectifs des différentes phases de l'optimiseur (calcul de la matrice de Gram, solveur quadratique), ansi que certaines techniques nouvelles (filtrage des pièces inutiles). Il reste également à traiter les inégalités dans le primal comme elles devraient l'être: par l'introduction des contraintes $\lambda = \hat{\lambda} + d \geq 0$ dans (13) ou (21), et non par pénalisation de ces contraintes via un groupe fictif.

En revanche, la décomposition spatio-temporelle (problème X) accumule toutes les difficultés. La présente étude n'a fait que révéler ces difficultés, et suggérer les remèdes. Plus d'expérimentation est nécessaire pour conclure. Une façon simple d'entamer cette expérimentation serait par exemple d'appliquer la décomposition croisée aux maquettes utilisées pour le problème N.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803